Subconjuntos Mínimos de Corrección para explicar características muertas en Modelos de Líneas de Productos. El caso de los Modelos de Características

Luisa Rincon
Universidad Nacional de Colombia
Medellín, Colombia
lufrinconpe@unal.edu.co

Gloria Giraldo
Universidad Nacional de Colombia
Medellín, Colombia
glgiraldog@unal.edu.co

Raul Mazo
CRI, Universidad Paris 1 Panthèon Sorbonne
Paris, Francia
raul.mazo @ univ-paris1.fr

Camille Salinesi
CRI, Universidad Paris 1 Panthèon Sorbonne
Paris, Francia
camille.salinesi @ univ-paris1.fr

Daniel Diaz
CRI, Universidad Paris 1 Panthèon Sorbonne
Paris, Francia
daniel.diaz@ univ-paris1.fr

Resumen

Aprovechar los beneficios que ofrecen las líneas de productos depende, entre otros aspectos, de la calidad de los modelos que representan cada línea de productos. Una parte de la calidad consiste en asegurar que los Modelos de Líneas de Productos (MLPs) se encuentran libres de defectos. Un tipo de defecto de los MLPs son las características muertas, ellas son elementos reutilizables que no están presente en ningún producto configurado a partir del MLPs. Cuando las características muertas aparecen, quien crea los MLPs necesita herramientas que le permitan identificar por qué se presentan las características muertas y cómo podría corregirse el modelo. Sin embargo, aunque muchos trabajos en la literatura identifican características muertas, pocos explican por qué se originan o lo explican de manera incompleta. En este artículo se propone un nuevo método para explicar por qué se presentan características muertas en un MLP expresado con la notación modelos de características. Nuestra explicación consiste en identificar diferentes subconjuntos de elementos que podrían ser modificados para corregir el modelo cada que se presente una característica muerta. Esta explicación ofrece al modelador información completa sobre cómo corregir el modelo para cada característica muerta encontrada.

Palabras claves; Modelos de Líneas de Productos, Explicación de Defectos, Características Muertas, Programación e Ingeniería de Software

Abstract

Take advantage of benefits offered by the paradigm of product lines depends, among other things, of quality of models that represent each product line. An important facet of quality is ensuring that the Product Line Models (PLMs) are free of defects. One type of defect of the PLMs are dead features, they are reusable elements not present in any valid product of the product line. When dead features appear, feature modeler need tools that help him to identify why this defect occurred and how he could correct the model. However, even if several works exist in literature for identifying dead features, few of them explain why dead features appear or they provide incomplete explanations. In this paper we propose a new method to explain why each dead feature occurs in PLMs expressed with the notation of feature models. Our explanation consists of identifying different subsets of elements that could be modified to correct the model for each dead feature. This explanation provides the modeler complete information on how correcting each dead feature.

Keywords; Product Line Models, Defect Explanations, Dead Features, Programming and Software Engineering.

I. INTRODUCCIÓN

Las líneas de productos son un paradigma que permite administrar eficientemente un conjunto de productos con elementos comunes y variables que pertenecen a un dominio en particular. Este paradigma ofrece beneficios como la reutilización, la disminución de errores y la disminución de tiempos y costos de producción. Empresas como Nokia y Hewlett Packard[1] han utilizado exitosamente este paradigma para la gestión de celulares e impresoras respectivamente.

Los beneficios propuestos para las líneas de productos pueden ser extensibles al software, pues en el desarrollo de software es necesario administrar la reutilización y la variabilidad. Las Líneas de Productos de Software [1] son el paradigma de las líneas de productos aplicado al desarrollo de software.

Todos los productos que pueden crearse a partir de una línea de productos son representados de manera implícita usando modelos de líneas de productos (MLPs). La notación de modelos de características, en inglés *Feature Models*, es una de las notaciones que se utiliza para hacer esta representación. Para aprovechar los beneficios ofrecidos por las líneas de productos se requiere que los MLPs se encuentren libres de defectos. Sin embargo, a medida que aumenta la complejidad de la línea de productos, es posible que quienes diseñan los modelos de características introduzcan sin intención defectos en los modelos. Las características muertas son un tipo de defecto que se origina cuando en el modelo de características existen elementos que no pueden estar presentes en ningún producto válido de la línea de productos [2], [3]. Este defecto es una contradicción entre lo que se desea representar y lo que el modelo de características realmente representa. Por ejemplo,

1 http://www.sei.cmu.edu/productlines/casestudies/?location=secondary-nav&source=10198

en la Figura 1, la característica `Mexapixeles` es una característica muerta. La sección IV presenta en detalle este defecto.

Diferentes trabajos se han propuesto en la literatura para identificar automáticamente cuáles son las características muertas de un modelo de características [4–13]. Sin embargo, pocos trabajos ofrecen alguna explicación sobre el origen de las características muertas en los modelos de características, o sobre cómo podría corregirse este defecto [9], [10]. De hecho ofrecer explicaciones sobre los defectos de los modelos de características es aún un campo abierto de investigación [14].

Cuando se conoce que una característica es muerta pero no se entiende por qué ocurrió, es necesario inspeccionar manualmente el modelo de características para decidir qué cambios son necesarios para solucionar el defecto. Sin embargo, hacer esta operación manualmente es tan complicado como encontrar el defecto mismo [5], [9], [11]En este trabajo proponemos un método que explica, para un modelo de características con características muertas, por qué se originó cada característica muerta. Una característica muerta es un requerimiento o componente reutilizable de la línea de productos que no puede ser usado en ningún producto configurado a partir del MLPs. La explicación consiste en presentar, para cada característica muerta identificada, los subconjuntos de relaciones que podrían modificarse en el modelo de características para corregir el defecto (Subconjuntos Mínimos de Corrección). Sin embargo, la verificación de que los subconjuntos de corrección identificados no generen otros defectos en el modelo es un trabajo futuro. Ilustramos la propuesta con la aplicación de nuestro método a un modelo de características de teléfonos celulares adaptado del modelo de características propuesto en [15].

La organización de este artículo es la siguiente: En la siguiente sección se presenta el marco conceptual de la investigación. Luego, en la sección III se presenta el método propuesto. En la sección IV se presentan los resultados obtenidos al aplicar nuestra propuesta al caso de estudio y se discuten los resultados preliminares. En la sección V se presenta la evaluación preliminar realizada. En la sección VI se presentan los trabajos relacionados con nuestra propuesta. Finalmente, en la sección VII se presentan las conclusiones y el trabajo futuro.

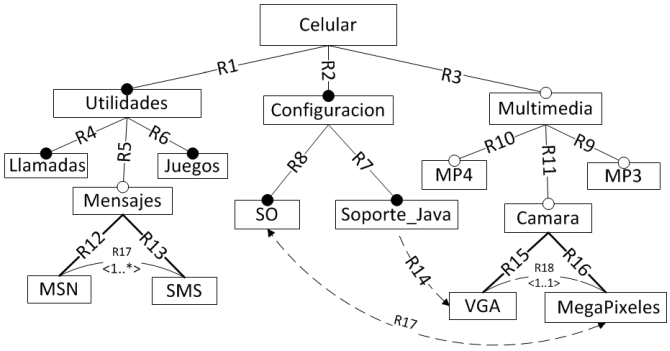

Figura 1. Modelo de características de teléfonos celulares. Versión adaptada del modelo propuesto en [15]

II. CONCEPTOS PRELIMINARLES

A. *Modelos de características*

La notación de modelos de características es utilizada para representar los elementos comunes y variables de un domino en particular. Esta notación además presenta las decisiones de *trade-off* que se deben tomar para crear un producto válido en el dominio que describe una línea de productos [2] .

En un modelo de características, cada característica corresponde a un elemento distintivo para el usuario y cada modelo tiene una única característica raíz que estará presente en todos los productos que hagan parte de la línea de productos [2].

Las relaciones que pueden ser usadas para unir entre si las características de un modelo de características son:

Obligatoria: La característica B debe ser incluida en todos los productos que tengan la característica padre A y viceversa. La relación *R1* de la Figura 1 es obligatoria, por lo tanto la característica `Utilidades` debe ser incluida en todos los productos que tengan la característica `Celular`. Otras relaciones obligatorias del modelo de la Figura 1 son `R2, R4, R6, R7 y R8.`

Opcional: La característica B puede o no ser incluida en todos los productos que tengan la característica padre A. Sin embargo, si la característica B es incluida en un producto, la característica A también debe ser incluida en el mismo producto. La relación `R3` de la Figura 1 es opcional, por lo tanto la característica `Multimedia` puede o no ser incluida en los productos que tengan la característica `Celular.` Otras relaciones opcionales del modelo de la Figura 1 son `R9, R10, R11`.

Requerida: La característica B debe ser incluida en todos los productos que tengan la característica A. La relación `R14` de la Figura 1 es requerida, por lo tanto la característica `VGA` es requerida por la característica `Soporte_Java.`

Exclusión: La característica A y B no pueden ser seleccionadas al mismo tiempo en un producto. La relación `R17` de la Figura 1 es de exclusión, por lo tanto la característica `Megapixeles` y `SO` no pueden estar simultáneamente en un mismo producto de la línea de productos.

Cardinalidad grupal: Representa el mínimo y máximo número de características que un producto puede tener cuando la característica padre es incluida en un producto. Si al menos una característica hija de la cardinalidad es incluida en un producto, la característica padre debe también ser incluida. La relación `R17` de la Figura 1 es una cardinalidad grupal que exige para todo producto mínimo una de sus dos características hijas y máximo las dos características hijas. La relación `R18` también es un cardinalidad grupal del modelo de la Figura 1.

La Figura 1 es un modelo de características que representa una línea de productos de teléfonos celulares permite crear 16 diferentes tipos de celulares. Para ilustrar nuestra propuesta, adaptamos este modelo del propuesto en [15] al simplificarlo e introducirle relaciones que volvían la característica `Megapixeles` una característica muerta. En las siguientes

secciones se explicará con detalle cuáles relaciones generan este defecto y cómo podría corregirse.

B. *Representación de los modelos de características como programas de restricciones*

Cuando se transforma un modelo de características a un programa de restricciones las características del modelo se convierten en variables del programa de restricciones. El dominio de las variables corresponde a valores enteros de cero en adelante y las relaciones entre las características del modelo se escriben como restricciones entre las variables. En [16] los autores proponen un conjunto de reglas para transformar modelos de características en programas de restricciones. La Figura 2 presenta el modelo de características de la Figura 1 como un programa de restricciones en SWI-Prolog[2]. Este programa de restricciones fue obtenido al aplicar las reglas de transformación propuestas en [16] al modelo de características de la Figura 1. La línea 3 representa las variables, la línea 4 representa el dominio de las variables y las líneas de la 5 a la 21 son restricciones que representan las relaciones del modelo.

Un modelo de características no tiene características muertas cuando el *solver* que analiza el correspondiente programa de restricciones puede asignarle el valor de 1 a cada una de las variables del programa [7]. Cada asignación de valores a las variables corresponde a un producto que puede ser creado a partir de la línea de productos. Uno ("1") significa que el producto tiene la característica y cero ("0") significa que no la tiene.

Por ejemplo, a partir del programa de restricciones de la Figura 2, puede crearse un celular con las siguientes características:

```
Celular=1, Utilidades=1, Llamadas=1,
Mensajes=0, MSN=0, SMS=0, Juegos=1,
Configuración=1,SO=1,Soporte_Java=1,
Multimedia=1, MP3=0, MP4=0, Camara=1,
VGA=1, Mexapixeles=0
```

Este celular tiene todas las características que componen el modelo de características con excepción de las características `Mensajes`, `MSN`, `SMS` `MP3`, `MP4` y `Megapíxeles`.

```
(1) :-use_module(library(clpfd)).
(2) productline(L):-
(3) L=[Celular,Utilidades,LLamadas,Mensajes,MSN,
    SMS,Juegos,Configuracion,SO,Soporte_Java,
    Multimedia,MP3,MP4,Camara,VGA,Mexapixeles],
(4) L ins 0..1,
(5) Celulares #= 1,
(6) Celulares#<==>Utilidades,
(7) Utilidades#<==>LLamadas,
(8) Utilidades#<==>Juegos,
(9) Celulares#<==>Configuracion,
(10) Configuracion#<==>SO,
(11) Configuracion#<==>Soporte_Java,
(12) Celulares#>=Multimedia,
(13)Multimedia#>=MP3,
(14)Multimedia#>=MP4,
(15)Multimedia#>=Camara,
(16)1 * Camara #= Mexapixeles + VGA,
(17)Utilidades#>=Mensajes,
(18)1 * Mensajes #=< SMS + MSN,
(19)SMS + MSN #=< 2 * Mensajes,
(20)Soporte_Java#==>VGA,
(21)SO * Mexapixeles #=0,
(22)label(L).
```

Figura 2. Modelo de características de teléfonos celulares expresado como un programa de resricciones en SWI-Prolog

[2] http://www.swi-prolog.org/

C. *Subconjunto Mínimo de corrección (MCS)*

El Subconjunto Mínimo de Corrección ,en inglés *Minimal Correction Subset* (*MCS*), es un concepto utilizados para el análisis de sistemas de restricciones insatisfacibles [17–19]. Un MCS es un subconjunto de restricciones M que pertenecen a un sistema insatisfacible de restricciones tal que:

$$C-M \text{ es satisfacible} \wedge$$
$$\forall Ci \in M,\ C - (M - \{ Ci \}) \text{ es satisfacible}$$

Lo anterior quiere decir que en el MCS sólo se encuentran las restricciones que al eliminarse vuelven el sistema de restricciones satisfacible. Si existe más de un elemento que puede eliminarse para corregir el defecto, entonces existe más de un MCS. La unión de todos los MCS corresponde a los subconjuntos mínimos de corrección, en inglés *Minimal Correction Subsets* (MCSes) [19].

III. PROPUESTA

Nuestra propuesta se basa en el análisis de sistemas de restricciones insatisfacibles propuesto en [18], [19] y en la utilización de programación de restricciones para analizar modelos de características propuesto en [7], [20], [21].

De una parte, en [7], [20], [21] los autores explican que es posible transformar los modelos de características a programas de restricciones que pueden ser interpretados por un *solver*. Desde el punto de vista de las restricciones, un modelo de características con características muertas es un conjunto de restricciones para las que no se puede satisfacer que la característica muerta tome el valor de uno. Por otra parte, en sistemas de restricciones insatisfacibles, la identificación de los MCSes sirve para identificar las restricciones que deben modificarse para volver el sistema de restricciones satisfacible.

Nuestra propuesta es usar la programación por restricciones y el concepto de los MCSes para explicar las características muertas en los modelos de características.

La Figura 3 presenta una visión general de nuestra solución. Para iniciar, se recibe como entrada un modelo de características expresado como un programa de restricciones en SWI-Prolog. Luego usamos una herramienta que realiza la verificación automática de modelos de variabilidad (VariaMos [22]) para identificar las características muertas del modelo analizado. Después, para cada característica muerta identificada con VariaMos se identifican los Subconjuntos Mínimos de Corrección (MCSes) que explican la característica muerta. Finalmente se presentan los MCSes identificados como la explicación del origen de la característica muerta analizada. Estos MCSes son las restricciones que deben ser modificadas para corregir la característica muerta analizada.

En nuestro contexto, cada MCS corresponde al subconjunto mínimo de relaciones del modelo que podrían modificarse para corregir la característica muerta analizada.

Cada subconjunto debe ser mínimo pues no debe contener relaciones que al modificarse no corrijan la característica muerta analizada. Además para quien corrige una característica muerta no le es de utilidad que la explicación del defecto sea

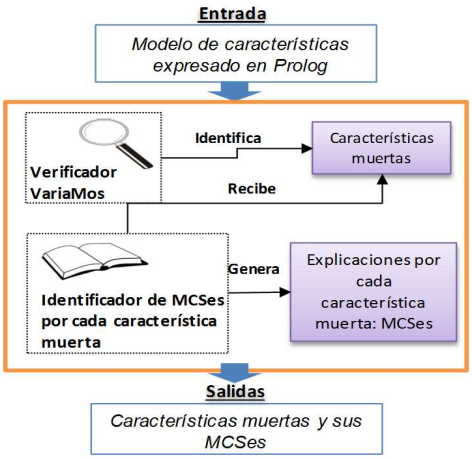

Figura 3. Proceso general propuesto para explicar las características muertas en los modelos de características

que todas las relaciones deben ser modificadas para corregir el defecto [23]. La identificación de los MCSes se realiza de la siguiente manera:

1) Se vuelve insatisfacible el sistema de restricciones que representa el modelo de características. Para ello se adiciona al conjunto de restricciones en Prolog una restricción adicional en la que la característica muerta a analizar debe tener el valor de 1. Por ejemplo, para el modelo de características de teléfonos celulares, se adiciona la restricción `Megapixeles#=1` *al conjunto de restricciones de la Figura 2.*

2) Se modifican progresivamente restricciones, con excepción de la restricción adicionada en el punto anterior, hasta que el conjunto de restricciones que representa el modelo de características vuelva a ser satisfacible. Cada restricción que al eliminarse hace que el conjunto de restricciones restantes sea satisfacible pertence a un MCS. Para las relaciones obligatorias la modificación consiste en volverlas opcionales. Para el resto de relaciones, la modificación consiste en eliminarlas del conjunto de restricciones que representan el modelo de características analizado.

Para encontrar todos los MCS, primero se modifican de una en una las restricciones del conjunto de restricciones original. En adelante se modifican de dos en dos las restricciones del conjunto original de restricciones y así sucesivamente. Para cada modificación se inicia siempre desde el conjunto de restricciones resultante del paso 1 anteriormente descrito.

Con el objetivo de garantizar que el conjunto de corrección es mínimo, una vez se identifica que al eliminar una restricción dada, el conjunto de restricciones se vuelve satisfacible, esta restricción no se tiene en cuenta en ningún otro subconjunto de restricciones a eliminar.

Nuestra propuesta es semiautomática pues debe realizarse manualmente la transformación del modelo de características a un programa de restricciones en Prolog utilizando las reglas propuestas en [16]. Adicionalmente, por cada característica muerta debe realizarse manualmente la adición de la restricción del paso 1 al conjunto de restricciones que representan el modelo de características. De ahí en adelante la identificación de los MCSes es automática. La implementación fue realizada utilizando Java y la librería de SWI-Prolog que permite ejecutar desde Java programas Prolog.

IV. Analisis de Resultados y Discusión

La Tabla 1 presenta los MCSes encontrados al aplicar nuestra propuesta al ejemplo de la Figura 1. Cada MCS contiene la menor cantidad posible de características que deben modificarse simultáneamente para corregir la característica muerta *Megapixeles.* La primera columna de la tabla presenta el identificador de cada MCS. La segunda columna presenta las relaciones del modelo de características que hacen parte de cada MCS. La tercera columna presenta las líneas del programa de restricciones de la Figura 2 que representan las relaciones que hacen parte del MCS. Finalmente, la cuarta columna presenta la cantidad de elementos que tiene cada MCS.

Tabla 1 Subconjuntos mínimos de corrección para la característica muerta Megapixeles del modelo de características de teléfonos celulares

Explicación	Relaciones del modelo (Figura 1)	Restricción en el programa de restricciones (Figura 2)	Cantidad de elementos del MCS
MCS1	R2	Línea 9	1
MCS2	R8,R7	Línea 10, Línea 11	2
MCS3	R8,R14	Línea 10, Línea 20	2
MCS4	R8,R18	Línea 10, Línea 16	2

Cuando se interpreta cada MCS, se encuentra que para corregir la característica muerta `Megapixeles` quien debe corregir el modelo de características podría: (i) modificar la relación obligatoria `R2`; (ii) modificar las relaciones obligatorias `R8` y `R7`; (iii) modificar la relación obligatoria `R8` y la relación requerida `R14`; o (iv) modificar la relación obligatoria `R8` y la relación de exclusión `R18`.

Como se puede observar en la Tabla 1, para un modelo de características con 16 características y 18 relaciones, nuestra propuesta encuentra 4 MCS. Esta variedad de posibilidades genera preguntas como ¿Cuál de todas las correcciones es mejor? ¿Es preferible la corrección que implique la menor cantidad de cambios en el modelo?, ¿Es seguro que al aplicar estas modificaciones no se generarán nuevos defectos en el modelo de características?

Quienes desarrollan los modelos de características, son en nuestra opinión, quienes pueden responder las dos primeras preguntas y decidir cuál de los MCS identificados es la mejor corrección. Esta decisión puede relacionarse por ejemplo con el MCS que implique la menor cantidad de cambios, o el MCS que no implique modificar alguna relación del modelo de características analizado. Por esa razón, nuestra aproximación, a diferencia de lo propuesto en [9], no toma como único criterio para identificar los MCS la menor cantidad de cambios en el modelo, sino que presenta todos los MCS que pueden ser

identificados a partir de la modificación sistemática de las restricciones que representan el modelo de características.

En cuanto a si al aplicar estas correcciones se puede garantizar que no se generan nuevos defectos, nuestra propuesta no contempla esa verificación adicional y la propone como un trabajo futuro.

V. EVALUACIÓN PRELIMINAR

Como una primera evaluación de la precisión de los resultados obtenidos, aplicamos nuestra propuesta a 5 modelos con características muertas creados con BeTTy [24], un generador aleatorio de modelos de características. De los 5 modelos generados, 3 modelos tenían 5 características y 2 modelos tenían 15 características. Los modelos tenían un número pequeño de características para facilitar la revisión manual de los resultados obtenidos. Sin embargo, la escalabilidad de la propuesta debe ser evaluada en trabajos futuros.

Los modelos utilizados para las pruebas pueden ser descargados de Internet[3] en formato *.pl* para SWI-Prolog y en formato *.splx* para ver el modelo de características en un formato bien conocido para representar modelos de características[4].

Una parte de nuestros resultados fueron comparados con las explicaciones obtenidas usando FaMa [25]. FaMa identifica el subconjunto mínimo de relaciones que implica la menor cantidad de cambios y que al modificarse podría corregir la característica muerta. La propuesta presentada en este artículo permite identificar los mismos subconjuntos de FaMa y adicionalmente identifica todos los otros subconjuntos mínimos de relaciones que también podrían ser modificados para corregir la característica muerta.

Posteriormente hicimos una inspección manual con el fin de verificar que al modificar las relaciones que hacían parte de cada MCS se corregía la característica muerta analizada.

Los resultados de la evaluación preliminar confirman la precisión de nuestra propuesta. Todos los MCS obtenidos explican la característica muerta analizada. Además, cada MCS es un conjunto mínimo de modificaciones que podrían hacerse en cada modelo para corregir cada característica muerta analizada.

VI. TRABAJOS RELACIONADOS

Diferentes trabajos se ha propuesto para identificar automáticamente características muertas (y otros defectos) en modelos de características [4–13], [22]. Sin embargo, ninguno de estos trabajos explica por qué se producen las características muertas identificadas. En ese sentido, nuestra propuesta podría ser utilizada como un complemento a los trabajos de verificación y análisis automático de modelos de características que ya existen en la literatura.

Trinidad *et al.* [9] presenta un método automático para identificar y explicar características muertas en modelos de características. En este trabajo los autores transforman el modelo de características en un problema de diagnóstico y luego, al igual que nuestra propuesta, resuelven un problema de satisfacción de restricciones para explicar las características muertas del modelo de características analizado. La propuesta fue automatizada en FaMa [25] un *plugin* de eclipse para análisis automático de modelos de características. En su trabajo los autores presentan como explicación para el origen de cada característica muerta algunas de las relaciones que podrían modificarse para corregir este defecto. Sin embargo, en este trabajo los autores solamente identifican el mínimo conjunto de relaciones que deben ser modificadas, y no exploran otros subconjuntos de relaciones que al modificarse también podrían corregir la característica muerta analizada.

Nuestra propuesta a diferencia del trabajo de Trinidad et *al*. [9], considera que pueden existir otras posibilidades de corrección de más utilidad para quien analiza el modelo que no implican necesariamente una mínima cantidad de cambios. Por esa razón, nuestro trabajo, como explicación del defecto, identifica todos los subconjuntos de relaciones que corrigen la característica muerta analizada y pueden identificarse a partir de la modificación sistemática de las restricciones que representan el modelo de características. Es de notar que aunque con nuestra propuesta puede identificarse más de una posibilidad de corrección, todos los subconjuntos contienen información diferente e involucran solamente las relaciones que realmente podrían ser modificadas para corregir el defecto.

En un trabajo posterior [10] Trinidad et *al*. proponen utilizar lógica abductiva para explicar porque una característica dada es una característica muerta. Sin embargo, los autores no presentan ningún algoritmo ni detalle que permita implementar su propuesta.

VII. CONCLUSIONES

En este trabajo usamos los Subconjuntos Mínimos de Corrección para explicar las características muertas identificadas en los modelos de características. La explicación consiste en presentar al usuario que analiza el modelo de características todos los subconjuntos mínimos de relaciones del modelo de características que podrían ser modificados para corregir cada característica muerta.

A diferencia de los trabajos encontrados en la literatura, nuestra propuesta explica cada característica muerta con la identificación de un amplio número de subconjuntos mínimos de corrección. Esta información le da a conocer a quien corrige el modelo de características las opciones disponibles para corregir el defecto, lo que le podría facilitar el proceso de corrección. Como indican en [2], [6], [9], [26], [27] toda información que permita guiar el proceso de corrección ofrece un ahorro de tiempo y costo en el desarrollo de la línea de productos. Conocer rápidamente el subconjunto de relaciones del modelo que podrían modificarse para corregir el defecto da agilidad al proceso de corrección y permite que quienes diseñan los modelos de características puedan aprovechar el

3 https://sites.google.com/site/raulmazo/

4 http://www.splot-research.org/

tiempo en hacer una buena representación del dominio de la línea de productos, más que en corregir defectos en los modelos.

Esperamos que al automatizar completamente nuestra propuesta esta pueda incorporarse a las herramientas de análisis automático de modelos de características como VariaMos [22] o FaMa [25]. De esta manera el proceso de verificación y corrección de defectos en modelos de características será de más utilidad cuando se utilizan los modelos de características a nivel industrial.

Estamos trabajando en extender nuestra solución para: (i) identificar de manera totalmente automatizada para cada característica muerta las relaciones que podrían modificarse para corregir el defecto; (ii) explicar otros tipos de defectos en modelos de características; (iii) garantizar que los MCS propuestos corresponden a relaciones que al ser modificadas no generen nuevos defectos en los modelos de características; (iv) explicar los defectos de los modelos de líneas de productos de una manera genérica que sea independiente de la notación en la que está expresada el modelo.

AGRADECIMIENTOS

El trabajo de investigación presentado en este artículo fue desarrollado durante la pasantía de investigación de la maestría en Ingeniería-Ingeniería de Sistemas parcialmente financiado por la Universidad Nacional de Colombia sede Medellín y el Centre de Recherche en Informatique (CRI) de la Universidad Paris 1 Panthéon Sorbonne que se desarrolló desde finales de diciembre del 2012 a finales de marzo del 2013.

REFERENCIAS

[1] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns, 1st ed. Addison-Wesley Professional, 2001.

[2] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, “Feasibility Study Feature-Oriented Domain Analysis (FODA). Technical Report,” 1990.

[3] P. Trinidad, D. Benavides, and A. Ruiz-Cortés, “Isolated Features Detection in Feature Models,” in Proceedings of Conference on Advanced Information Systems Engineering (CAiSE 2006), 2006, vol. 01, pp. 1–4.

[4] D. Batory, “Feature models, grammars, and propositional formulas,” Software Product Lines, pp. 7–20, 2005.

[5] A. Osman, S. Phon-Amnuaisuk, and C. Kuan Ho, “Knowledge Based Method to Validate Feature Models,” in First International Workshop on Analyses of Software Product Lines, 2008, pp. 217–225.

[6] A. Osman, S. Phon-Amnuaisuk, and C. Kuan Ho, “Investigating Inconsistency Detection as a Validation Operation in Software Product Line,” in Studies in Computational Intelligence, vol. 253, 2009, pp. 159–168.

[7] C. Salinesi and R. Mazo, “Defects in Product Line Models and how to Identify them,” in Software Product Line - Advanced Topic, InTech., A. Elfaki, Ed. 2012, pp. 1–40.

[8] T. Thüm, C. Kastner, F. Benduhn, J. Meinicke, and Saak, “FeatureIDE: An extensible framework for feature-oriented software development,” Science of Computer Programming, 2012.

[9] P. Trinidad, D. Benavides, A. Duran, A. Ruiz-Cortes, and M. Toro, “Automated Error Analysis for the Agilization of Feature Modeling,” Journal of Systems and Software, vol. 81, no. 6, pp. 883–896, 2008.

[10] P. Trinidad and A. Ruiz-Cortes, “Abductive Reasoning and Automated Analysis of Feature models: How are they connected,” in Proceedings of the Third International Workshop on Variability Modelling of Software-Intensive Systems, 2009, pp. 145–153.

[11] H. Wang, Y. Li, J. Sun, H. Zhang, and J. Pan, “Verifying feature models using OWL,” Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, no. 2, pp. 117–129, Jun. 2007.

[12] W. Zhang and H. Zhao, “A Propositional Logic-Based Method for Verification of Feature Models,” in Proceedings of the 6th International Conference on Formal Engineering Methods (ICFEM’04), 2004, pp. 115–130.

[13] T. Von der Massen and H. Lichter, “Deficiencies in Feature Models,” in Workshop on Software Variability Management for Product Derivation - Towards Tool Support, 2004.

[14] R. Mazo, “A Generic Approach for Automated Verification of Product Line Models,” Ph.D.thesis.Paris 1 Panthéon – Sorbonne University, Paris, France, 2011.

[15] S. Segura, “Automated Analysis of Feature Models Using Atomic Sets,” in Proceedings of the First Workshop on Analyses of Software Product Lines (ASPL08), 2008.

[16] R. Mazo, C. Salinesi, D. Diaz, and A. Lora-Michiels, “Transforming Attribute and Clone-Enabled Feature Models Into Constraint Programs Over Finite DomainsMazo, R., Salinesi, C., Diaz, D., & Lora-Michiels, A. (2011). Transforming Attribute and Clone-Enabled Feature Models Into Constraint Programs Over Finite,” in Proc. of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2011, pp. 188–199.

[17] J. Bailey and P. J. Stuckey, “Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization,” in Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages (PADL’05). Lecture Notes in Computer Science, 2005, pp. 174–186.

[18] M. H. Liffiton and K. A. Sakallah, “Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints,” Journal of Automated Reasoning, vol. 40, no. 1, pp. 1–33, 2008.

[19] M. H. Liffiton, “Analyzing Infeasible Constraint Systems,” Ph.D.thesis.University of Michigan, 2009.

[20] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, “Automated reasoning on feature models,” in Proceedings of the 17th international conference on Advanced Information Systems Engineering, 2005, vol. 3520, pp. 491–503.

[21] D. Benavides, A. Ruiz-Cortes, and P. Trinidad, “Using constraint programming to reason on feature models,” in Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEK’5), 2005, vol. 2005, pp. 677–682.

[22] R. Mazo, C. Salinesi, and D. Diaz, “VariaMos : a Tool for Product Line Driven Systems,” in Proceedings of the 24th International Conference on Advanced Informa- tion Systems Engineering (CAiSE Forum’12), 2012, no. June, pp. 25–29.

[23] D. Benavides and P. Trinidad, “A survey on the automated analyses of feature models,” Jornadas de Ingeniería, pp. 1–10, 2006.

[24] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés, “BeTTy: benchmarking and testing on the automated analysis of feature models,” in Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, 2012, pp. 63–71.

[25] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez, “Fama framework,” in Software Product Line Conference, 2008. SPLC’08. 12th International, 2008, vol. 81, no. 6, pp. 359–359.

[26] J. Sun, H. Zhang, Y. Fang, and L. H. Wang, “Formal semantics and verification for feature modeling,” in Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on, 2005, pp. 303–312.

[27] K. Lauenroth, A. Metzger, and K. Pohl, “Quality Assurance in the Presence of Variability,” in Intentional Perspectives on Information Systems Engineering, S. Nurcan, C. Salinesi, C. Souveyet, and J. Ralyté, Eds. Springer Berlin Heidelberg, 2010, pp. 319–333.